Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
Electronic Notes in Theoretical Computer Science 176 (2007) 69–87
www.elsevier.com/locate/entcs

# Parametric Performance Contracts: Non-Markovian Loop Modelling and an Experimental Evaluation

Heiko Koziolek[1] Viktoria Firus[2]

*Graduate School Trustsoft*
*University of Oldenburg*
*26121 Oldenburg, Germany*[3]

**Abstract**

Even with todays hardware improvements, performance problems are still common in many software systems. An approach to tackle this problem for component-based software architectures is to predict the performance during early development stages by combining performance specifications of prefabricated components. Many existing methods in the area of component-based performance prediction neglect several influence factors on the performance of a component. In this paper, we present a method to calculate the performance of component services while including influences of external services and different usages. We use stochatic regular expressions with non-Markovian loop iterations to model the abstract control flow of a software component and probability mass functions to specify the time consumption of internal and external services in a fine grain way. An experimental evaluation is reported comparing results of the approach with measurements on a component-based webserver. The evaluation yields that using measured data as inputs, our approach can predict the mean response time of a service with less than 2 percent deviation from measurements taken when executing the service in our scenarios.

*Keywords:* performance prediction, parametric performance contracts, service time distribution, software components, stochastic regular expressions, non-Markovian loops

## 1 Introduction

Despite the rapidly growing performance (i.e. time efficiency in terms of response time and throughput) of computer hardware, performance problems can still be observed in many large hardware/software systems today. One reason for these persisting problems is the ever growing complexity of software. Large software system have to deal with complex user requirements and are often designed with architectures that do not scale well, even with additional hardware.

[1] Email: heiko.koziolek@informatik.uni-oldenburg.de
[2] Email: viktoria.firus@informatik.uni-oldenburg.de
[3] This work is supported by the German Research Foundation (DFG), grant GRK 1076/1


doi:10.1016/j.entcs.2006.02.032

Component-based software systems [1] might offer major improvements for engineering software. Lots of research is directed at analysing the performance of component-based software architectures during early development stages. The aim is to predict the performance of a whole architecture by combining the performance specifications of single prefabricated components. Component developers shall provide these specifications in a parameterisable form, so that the software architects or component assemblers can feed them into tools to gain an estimation on the expected performance of their design. This way, design decisions regarding components shall be supported and component assemblers shall be enabled to compare functional equivalent components for their non-functional properties.

To specify the performance of a component, multiple external influences have to be considered, because components shall be third-party deployable [1]. Components may execute external services to provide their services, they can be deployed on different hardware, operating systems and middleware platforms. Furthermore, they interact with their environment by using limited resources of a system, and they are possibly executed by very different user groups. Any approach aiming at predicting performance properties of component based systems has to take all of these factors into account to be precise. Most of the existing approaches fall short in at least one of the mentioned factors [2].

In this paper, we extend our former approach for modelling the dependency of a component's performance to external services [3]. We also implicitly consider influences of different usage profiles by specifying transition probabilities on control flow branches and probability mass functions for the number of loop iterations. We use stochastic regular expressions [4] to construct a performance model based on service effect specifications for a component service [5]. An experimental evaluation is provided, as we have implemented our approach and applied it on an experimental webserver.

Our approach is *precise*, because we use discrete probability mass functions to model the time consumption of internal and external computations. It is *compositional*, since the result of our prediction is again a discrete probability mass function and can be used to model the performance of an external service for another component. Furthermore, the approach is *parametric*, because different probability mass functions can be used (for example depending on the underlying hardware) and the prediction can be adjusted to a different usage profile by changing transition probabilities and probability functions for loop iteration. Moreover, performance contracts for component services specified in the Quality of Service Modelling Language (QML) [6] can be checked with our method.

The contribution of this paper is twofold: First, we present a new concept for modelling loops with probability mass functions and introduce stochastic regular expressions. Second, we report on an experimental evaluation of our approach on a prototypical component-based software system and compare the predictions of our approach with measurements. As can be seen in our evaluation, the calculated probability mass functions differ only slightly from the actual measured values emphasizing the precision of our approach.

The paper is organised as follows: Section 2 introduces our modelling approach based on service effect specifications, describes the stochastic annotations and explains the calculations. Section 3 contains the description of our experimental evaluation and illustrates the results. Section 4 discusses related work and section 5 concludes the paper and outlines future work.

# 2 Modeling Component Performance

## *2.1 Service Effect Specifications*

Our model to describe the performance of a component is based on service effect specifications [5], which have been proven to be useful to model the influence of external dependencies on software components. When modelling the performance (i.e. response time, throughput) of a component, the component's calls to required services have to be taken into account, because they affect the performance perceived by the users of the component. A service effect specification models the calls of a provided service of a component to its required services. Usually, the specification is represented as an automaton to restrict possible sequences of calls. The transitions of the automaton model external calls and the states represent internal computation of the component. Thus, a service effect automaton can be considered as a control flow abstraction.

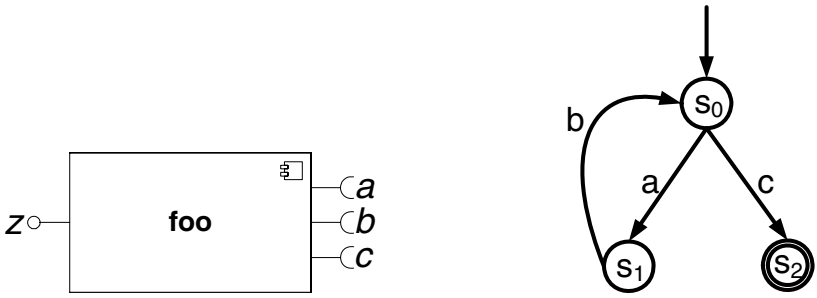


Fig. 1. Example component `foo` and corresponding service effect automaton for service `z`

Fig. 1 shows an example component `foo` in UML 2.0 notation on the left side, with the provided service `z` and the required services `a`, `b`, and `c`. The service effect automaton for service `z` is presented on the right side. It is a finite state machine describing all possible call sequences emitted by `z`. Service effect specification have to be provided by the component developer for component assemblers. Service effect specifications can be generated out of source code or derived from design documents.

As we also want to model the time consumption of the internal computations of a service, we first decompose the service effect automaton. For each state an additional transition and state is inserted and the transition is labelled with the name of the former state (Fig. 2). For example, a transition $s_2$ is added from the

former state $s_2$ to the new final state. This step is also necessary to ease the later computations.

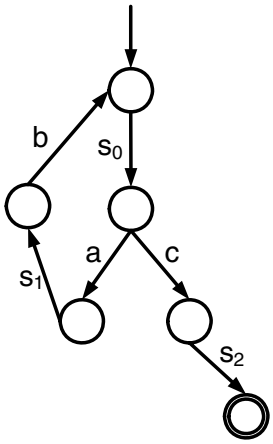


Fig. 2. Decomposed service effect automaton

Loops with the control flow are modelled by cycles in the service effect automaton. Cycles complicate the analysis, since they can be interconnected, include other cycles or have multiple entry points. We want to model loop iterations with probability functions, which will be described later, so we have to identify all cycles explicitly. Because of this, we convert the service effect automaton into a regular expression. Regular expressions are hierarchically structured and loops have a clear entry and exit point and can be identified by the Kleene star operator. Furthermore, regular expressions are well suited for the later computations, because it is possible to perform the calculation by traversing their abstract syntax tree. For the conversion, we use the GNFA-algorithm [7]. The regular expression for the above service effect automaton is:

$$(s_0\ \mathtt{a}\ s_1\ \mathtt{b})^*\ s_0\ \mathtt{c}\ s_2$$

### *2.2 Stochastical Annotations*

Service effect specifications were originally designed to calculate the requires protocol of a software component out of the provides protocol and to support interoperability checking between components [8]. To conduct performance analysis we need probabilities for the branches in the control flow, descriptions on the number of loop iterations and timing values attached to the elements of the regular expressions.

We describe stochastic regular expressions (SRE) (similar to [4]) in the following. Let $a$ be a terminal symbol from an alphabet $\Sigma$. The **syntax** of a SRE $R$ is recursively defined as:

$$R := a^{p_a} | R_1^{p_1} + R_2^{p_2} | R_1^{p_1} \cdot R_2^{p_2} | R^l$$

For the alternative $R_1^{p_1} + \ldots + R_n^{p_n}$, it must hold that $\sum_{i=1}^{n} p_i = 1$, the sum

of probabilities for the different alternatives at a branch in the control flow always equals one. The dot within a concatenation can be omitted.

For the loop construct $R^l$, $l$ is defined as a probability mass function over the number of loop iterations $n$ with

$$l : n \longmapsto p_n, n \in \mathbb{N}_0, 0 \leqslant p_n \leqslant 1, \exists N \in \mathbb{N}_0 : \forall i > N : l(i) = 0, \sum_{n=0}^{N} l(n) = 1$$

With this definition, we assign a probability to each number of loop iterations (e.g. it is possible to define that a loop is executed 3 times with a probability of 0,2 and 6 times with a probability of 0,8). Furthermore, the number of loop iterations is bounded, as we only allow probabilities greater than 0 for a limited number of iterations. This is a more practical approach to model loops than in classical Markov models [9]. There, loops are modelled with a probability $p$ of re-entering the loop and a probability $1-p$ of exiting the loop. This binds the number of loop iterations to a geometrical distribution, and it cannot be expressed for example that a loop is executed exactly $n$-times. It can not be expressed that a larger number of loop iterations is executed with a higher probability than a smaller one. But such situations can be found in many applications in practice, where a geometrical distribution on loop iterations is rather an exception. The problem of modelling loops with Markov models has also been stated by Doerner et. al. [10], who tackle the problem with a different approach. Other approaches models loops simply by providing mean iteration numbers [11]. This approach is also not favourable, because in reality, the number of loop iterations may differ heavily based on the usage of the component, and a mean value falls short to model this behaviour.

The **semantics** of the above syntax is defined as follows:

- Symbol ($a^{p_a}$): a is executed with probability $p_a$.
- Alternative ($R_1^{p_1} + R_2^{p_2}$): either $R_1$ is executed with probability $p_1$ or $R_2$ is executed with probability $p_2$. The total probability of the expression is $p_{alt} = p_1 + p_2$.
- Sequence ($R_1^{p_1} \cdot R_2^{p_2}$): first $R_1$ is executed, then $R_2$ is executed. The total probability of the expression is $p_{seq} = p_1 * p_2$.
- Loop ($R^l$):

$$\begin{cases} R & \text{is executed with probability} \quad l(1) \\ R \cdot R & \text{is executed with probability} \quad l(2) \\ \vdots & \\ \underbrace{R \cdot \ldots \cdot R}_{n} & \text{is executed with probability} \quad l(n) \end{cases}$$

The total probability of the expression is $\sum_{i=1}^{n} l(i) = 1$.

For the example regular expression from above, a possible SRE could be

$$(\mathtt{s}_0^{p_{s_0}} \ \mathtt{a}^{p_a} \ \mathtt{s}_1^{p_{s_1}} \ \mathtt{b}^{p_b})^l \ \mathtt{s}_0^{p_{s_0}} \ \mathtt{c}^{p_c} \ \mathtt{s}_2^{p_{s_2}}$$

with $p_{s_0} = p_a = p_{s_1} = p_b = p_c = p_{s_2} = 1,0$ and $l(7) = 1,0$.

To model the timing behaviour of the expression, a random variable $X_R$ is assigned to each input symbol $R$ as described in [3]. The random variable can be described by a probability mass function. It models the time consumption of a called service or an internal computation. In the following, we write $x_\alpha[n]$ for a probability mass function

$$x_\alpha : n \longmapsto p_n = P(X = \alpha n)$$

with sampling rate $\alpha$. The sampling rate $\alpha$ is the stepwidth for the intervals in the probability mass function and can be chosen for example as the greatest common divisor ($gcd$) of the interval's lengths: $\alpha = gcd\{[\![x_{i-1}, x_i]\!]\}_{i=1,2,\ldots,m-1}$.

For example, service $s_0$ from the stochastic regular expression above is assigned with a random variable $X_0$, whose probability mass function models how long $s_0$ is executed. The probability mass function is either the result of a computation, measured, or estimated (cf. section 2.4).

Using random variables instead of constant values allows a more fine grain performance prediction and is well suited to model internal and external time consumptions which are not fixed to constant values. They depend on a number of factors like the underlying hardware and middleware platform, the internal state of the component or inputs entered by users.

### *2.3 Computations*

The performance of a provided service is computed out of the annotated service effect specifications expressed as a stochastic regular expression. For the computations the abstract syntax tree of the regular expression is built and annotated with the probabilities, loop iterations functions, and random variables for time consumption. The tree is then traversed bottom-up until the resulting random variable for the root node (i.e. for the provided service) is computed. In the following, we explain the computation steps for the basic constructs sequence, alternative, and loop.

The time consumption for a **sequence** is the sum of the time consumptions for each expression. The probability mass function for a sequence can be computed as the convolution of the single probability mass functions:

$$x_{R_1 \cdot R_2}(n) = x_{R_1} \circledast x_{R_2}[n]$$

For an **alternative**, the time consumption is computed as the sum of the alternative paths weighted by the branch probabilities. The corresponding probability mass function is:

$$x_{R_1 + R_2}(n) = p_1 x_{R_1}[n] + p_2 x_{R_2}[n]$$

As we have specified a probability mass function for the number of loop iterations, the random variable for the **loop** is:

$$X_{R^l} = \begin{cases} X_R & \text{with probability} \quad l(1) \\ X_R + X_R & \text{with probability} \quad l(2) \\ \vdots & \\ \sum_{i=1}^{N} X_R & \text{with probability} \quad l(N) \end{cases}$$

with $N \in \mathbb{N}_0$ and $\forall i \geqslant N : l(i) = 0$. Thus the probability mass function for the loop has the following form:

$$x_{R^l}(n) = \sum_{i=1}^{N} l(i) \overset{i}{\underset{j=1}{\circledast}} x_R[n]$$

To compute the convolutions of the probability mass functions we make use of discrete Fourier transform as described in [3], where also the computational complexity of the approach is discussed in detail.

The probability mass function for the provided service `z` of the component `foo` from the example above has the form:

$$x_z(n) = \left( \sum_{i=1}^{7} l(i) \overset{i}{\underset{j=1}{\circledast}} (x_{s_0} \circledast x_a \circledast x_{s_1} \circledast x_b) \right) \circledast x_{s_0} \circledast x_c \circledast x_{s_2}[n]$$

### *2.4 Getting the necessary values*

To conduct the performance analysis with our approach, different inputs are needed. Service effect specifications can be obtained by analysing source code or design documents. For example, UML sequence diagrams might have been specified for some components and contain the information to construct a service effect specification.

The service effect specifications are annotated with transition probabilities and random variables for loop iterations. The component developer cannot provide these values by analysing the code of a component, because these values may depend on how users execute the components. For example, at an alternative in the service effect specification the control flow may take a different direction depending on the input parameter values of the service. Because of this, the values have to be provided by the system assembler, who has a certain expectation on how the anticipated users will use the component. An approach similar to the work by Hamlet et. al. [12] might be taken by the system assembler to obtain transition probabilities and loop iteration probabilities.

Furthermore, the service effect specification is annotated with random variables expressing the time consumption of external services (on transitions) and the time consumption of internal computations (on states). The needed probability functions might be derived from measurements (e.g. by benchmarking the component and the external services) or by estimating the values based on former experience, possibly supported by an approach like SPE [13].

It is still unclear, if our approach can be fully applied on existing black box components, for example by analysing byte code. Code annotations may have to be made to fully retrieve the needed information for our approach from source code. The direction of reengineering existing components to make them useful for our approach is part of our future research.

# 3 Experimental Validation

In the following, we provide a first initial experimental evaluation of our performance prediction approach. We compare measured data with calculated data assuming that the input data for the calculations are available. This assumption also needs to be tested in the future, as it is not clear if developers of component-based systems can obtain all the necessary data with the needed precision. The assumption could be tested with a controlled experiment involving students or with an experiment in an industrial setting, but this is beyond the scope of the validation described here.

The goal of our experimental evaluation was to analyse our performance prediction method from the viewpoint of the developer. We ask the following questions and use the described metrics to answers these questions:

- **Precision:** How precise are the calculation opposed to the measurements? As metrics we use descriptive statistics.
- **Sampling Rate:** What is the influence of the sampling rate of the probability mass functions to our calculations? We will use descriptive statistics and the time used for the calculations for different sampling rates to answer this question.
- **Efficiency:** How efficient can the calculation be performed? The time for the calculations will be used as a metric here. Furthermore, we will discuss the complexity of the calculations.

## *3.1 Subject of the Experiment: Webserver*

For our experiment, we applied the performance prediction approach on a component-based web server, which has been developed in our group for research purposes. The server has been designed for the .NET-platform and is implemented in C#. It is multi-threaded, as a dispatcher component spawns a new thread for each incoming request. Pages can be served either statically from harddisk, or they can be assembled dynamically by using the contents of a connected database (Microsoft SQL-server).

The webserver consists of 12 components, which are organised in a flexible architecture. Multiple components exist to handle different kinds of requests (e.g. HTTP requests) to the webserver. These components are organised in a Chain-of-Responsibility pattern [14], to easily allow extensions for other kinds of requests (e.g. FTP or HTTPS requests). We omit the full architecture of the server here and focus instead on predicting the performance of a single component, while taking connected components into account.

The `StaticFileProvider` component (Fig.3) provides an interface called `IHTTP-`

`RequestProcessor` to handle simple HTTP requests and retrieves files from harddisk. It requires the `IWebserverMonitor` and the `IWebserverConfiguration` interface to write log messages and to query the global webserver configuration. Furthermore, the `IRequestProcessorTools` interface is used to open and send files to the client.


Fig. 3. Component StaticFileProvider

The service effect automaton [15] of the service `HandleRequest` describes the abstract behaviour of the service and contains a transition for each external service called (Fig. 4). First, the path information is built for the requested file and a check is performed if the path actually exists. If a file is requested, the service writes a log entry and then starts to read the file from harddisk and transfer it to the client. If a directory is requested, the service first checks the webserver configuration (via the `IWebserverConfiguration` interface) for allowed standard file names (e.g. `index.htm`) and then checks in a loop, if one of the standard file names can be found in the requested directory. This file is retrieved and sent to the client. If the file can not be found or the default file name does not exist, an error message is sent to the client.

The regular expression corresponding to the service effect specification (Fig. 5) has been obtained by applying the GNFA algorithm [7]. The names of the transitions are abbreviated with the capital letter (e.g. BCP for BuildCompletePath). We omit the regular expression for the states of the service effect automaton here for the sake of brevity.

In the following, we define the independent, dependent and possible interfering variables of our experiment. *Independent variables* are not altered between measurements and calculations and define the context of our experiment. They are namely the hardware, the middleware platform, the deployment, the analysed component (`StaticFileProvider`) and the usage profile. To reduce the influence of the usage profile on the outcome of the experiment, we analysed three different scenarios for the service effect specification described above, which will be described later.

The *dependent variable* of our experiment is the metric we measured on the webserver, which was the response time for the different scenarios in milliseconds.

Possible *interfering variables* were active background processes, which could have distorted the measurements, caching of the webserver, time consumption for the output of log messages, the possible influence on the measurement approach to the measured results, garbage collection, just-in-time compiling etc. . We tried to keep the effect of the interfering variables as low as possible, by e.g. deactivating background processes, and implementing the measurement application as efficient as possible.

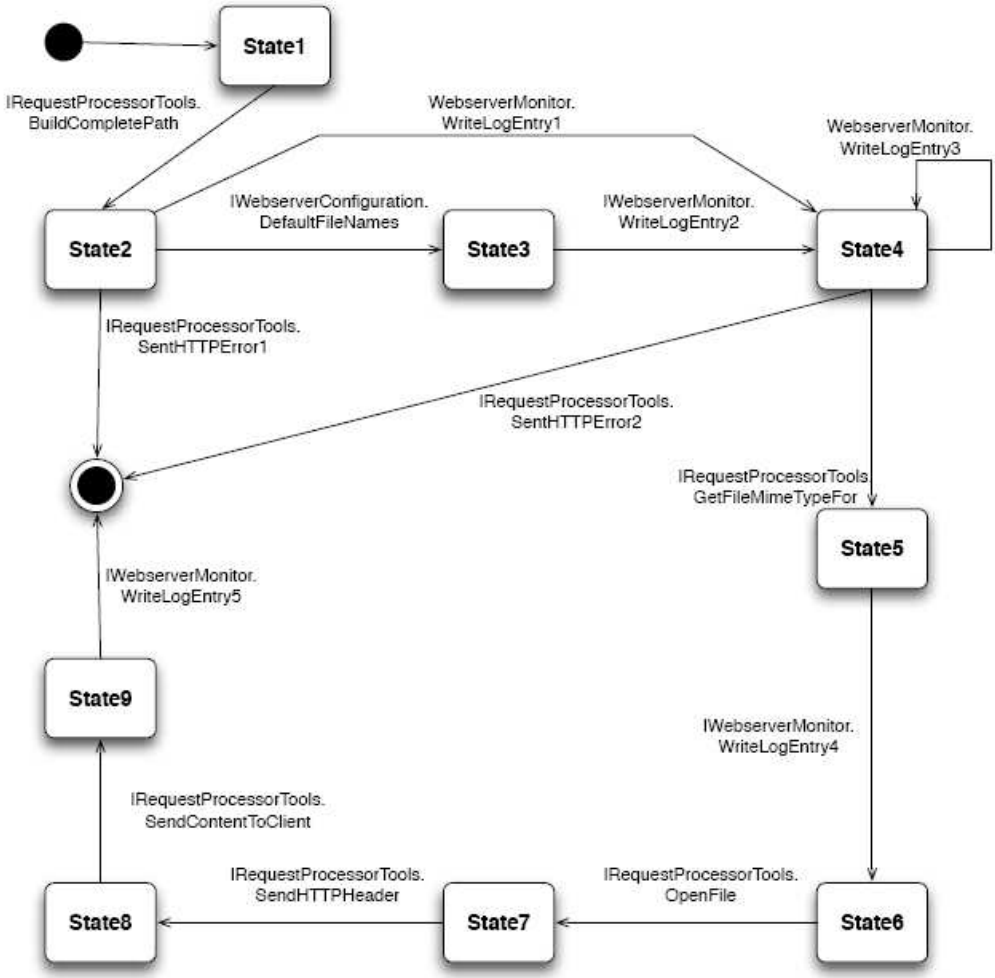


Fig. 4. `IHTTPRequestProcessor.HandleRequest`: Service Effect Automaton

```
BCP SHE1 +
(BCP GDF WL2 + BCP WL1) (WL3)* SHE2 +
(BCP GDF WL2 + BCP WL1) (WL3)* GFM WL4 OF SHH SCD WL5
```

Fig. 5. `IHTTPRequestProcessor.HandleRequest`: Regular Expression

### *3.2 Scenarios*

We applied our method on three different scenarios for the `HandleRequest` service of the StaticFileProvider component. The scenarios reflect the basic control flow constructs of sequence, alternative, and loop.

In *Scenario 1*, a 50 KByte HTML-file was requested from the webserver. As no directory was requested, no loop was executed in this scenario ($l_1(0) = 1.0$ and $l_1(i) = 0$, $\forall i \in \mathbb{N}$) and the control flow simply followed a sequence. Fig. 6 shows the stochastic regular expression for this scenario. Included are only the probabilities for alternatives to make the expression more readable, the probabilities for the other expression are all 1,0.

$$\begin{array}{c}(\texttt{BCP SHE1})^{0.0} + \\ (((\texttt{BCP GDF WL2})^{0.5}+(\texttt{BCP WL1})^{0.5})\ (\texttt{WL3})^{l_1}\ \texttt{SHE2})^{0.0}+ \\ (((\texttt{BCP GDF WL2})^{0.0}+(\texttt{BCP WL1})^{1.0})\ (\texttt{WL3})^{l_1}\ \texttt{GFM WL4 OF SHH SCD WL5})^{1.0}\end{array}$$

Fig. 6. Stochastic Regular Expression for Scenario 1 (Sequence)

*Scenario 2* involved two different request, triggering an alternative in the control flow. A 50 KByte HTML file and a subdirectory were requested alternately. The loop was not executed ($l_2(0) = 1.0$ and $l_2(i) = 0$, $\forall i \in \mathbb{N}$) as the file in the subdirectory was immediately found in this scenario (Fig. 7).

$$\begin{array}{c}(\texttt{BCP SHE1})^{0.0} + \\ (((\texttt{BCP GDF WL2})^{0.5}+(\texttt{BCP WL1})^{0.5})\ (\texttt{WL3})^{l_2}\ \texttt{SHE2})^{0.0}+ \\ (((\texttt{BCP GDF WL2})^{0.5}+(\texttt{BCP WL1})^{0.5})\ (\texttt{WL3})^{l_2}\ \texttt{GFM WL4 OF SHH SCD WL5})^{1.0}\end{array}$$

Fig. 7. Stochastic Regular Expression for Scenario 2 (Alternative)

*Scenario 3* contained the execution of the loop for 5 times, as a subdirectory was requested, and the webserver searched for 5 different file names before finding the file. Thus, the probability function for the loop iteration was $l_3(5) = 1.0$ and $l_3(i) = 0$, $\forall i \in \mathbb{N}_0 \backslash \{5\}$. Otherwise, the transitions were taken sequentially in this scenario, illustrated by Fig. 8.

$$\begin{array}{c}(\texttt{BCP SHE1})^{0.0} + \\ (((\texttt{BCP GDF WL2})^{0.5}+(\texttt{BCP WL1})^{0.5})\ (\texttt{WL3})^{l_3}\ \texttt{SHE2})^{0.0}+ \\ (((\texttt{BCP GDF WL2})^{0.0}+(\texttt{BCP WL1})^{1.0})\ (\texttt{WL3})^{l_3}\ \texttt{GFM WL4 OF SHH SCD WL5})^{1.0}\end{array}$$

Fig. 8. Stochastic Regular Expression for Scenario 3 (Loop)

### *3.3 Measurements and Calculations*

For measuring the scenarios, we implemented a monitoring framework for our webserver, using interceptors to decorate component interfaces with a measuring facility [14]. The response time of each service call was measured. The data was stored in memory during the measuring process and written to disk in an XML file format after the webserver was shut down. This way, we tried to remove the interfering influence of monitoring on the measurement results because of harddisk accesses.

The measurements were performed locally on a laptop with a Pentium-M processor 1.6 GHz, and 512 MB RAM with the webserver running on the .NET-platform. The requests for the scenarios were generated by a commercial web stress testing tool and subsequently repeated for one minute in each scenario. All requests were performed non-concurrently.

For the calculations, we used the measured data for single services as input to specify the random variables of time consumption for each service. The transition probabilities and loop iteration number were generated out of the measured data to ensure consistency between both approaches. During calculations, the abstract syntax tree of the regular expression is traversed and the time consumption of sequences, alternatives, and loops are computed bottom-up.

| | Mean | Std. Dev. | Max Y |
|---|---|---|---|
| Measurement ($\mu$s) | 2124,78 | 1387,57 | 1650 |
| Calculation ($\mu$s) | 2115,15 | 1253,44 | 1670 |
| Deviation (%) | 0,45 | 9,67 | 1,20 |

Table 1
Scenario 1 (Sequence): Descriptive Statistics

We used a sampling rate of 10 for each calculation, meaning that 10 values were combined from the probability mass functions to compute each value of the resulting function. Furthermore, we measured the duration for each calculation.

### *3.4 Results*

#### *3.4.1 Precision*

Fig. 9-11 show the probability mass functions for the response time (in $\mu$s) of each scenario of the `HandleRequest` service from the StaticFileProvider component. Measured data (dashed line) is compared with calculated data (dark line).

For the sequential execution of scenario 1, the calculated probabilities closely conform to the measured probabilities (observable in Fig. 9). The calculated line appears more smooth than the measured line because of the convolutions involved in the calculations. As a goodness of fitness test of two distribution functions, a $\chi^2$-test is commonly used in statistics [16]. The $\chi^2$-test with three degrees of freedom on the two distribution functions here yielded a too high value for $\chi^2$ to hold our defined significance level of 0,05. Thus, the deviation of both results was too high to confirm the same underlying probability function.

However, we would still argue that the results are useful because the deviation is small from a developer's perspective. The mean values of measurements and calculations only deviate by 0,45% and the standard deviation by 9,67% (Tab. 1). The maximum values of the probabilities are at 1650 $\mu$s for the measurements and at 1670 $\mu$s for the calculations, meaning that the most probable values only deviate by 1,2 percent in the response time. We also found that 80 percent of the values lie between 1500 and 2000 $\mu$s, both for measurements and calculation. For a performance analyst trying to predict the reponse time of components during early development stages with lots of still unstable informations, the precision should be adequate to make rough estimations and to support design decisions.

Scenario 2 involved an alternative in the control flow of the service effect specification. Like the sequential execution of scenario 1, the calculated probabilities resemble the measured probabilities closely, as can be observed in Fig. 10. Nevertheless, the hypothesis for the $\chi^2$-test had to be rejected as in scenario 1.

But like in scenario 1, the mean values and the standard deviation of measurement and calculations are very similar (Tab. 2). The maximum probabilities can be found at 1650 $\mu$s (measurements) and 1690 $\mu$s (calculations), yielding a deviation of 2,3 percent in response time. 75 percent of the probabilities of measurements and

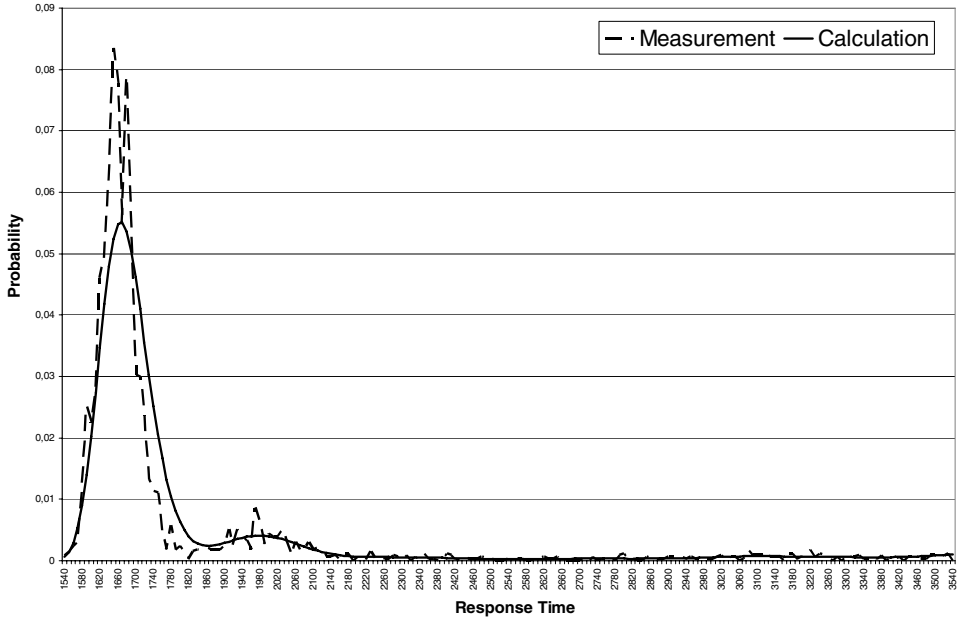


Fig. 9. Scenario 1 (Sequence): Probability Mass Functions

calculations lie between 1550 and 1920 $\mu$s. So the calculations are almost as precise as in scenario 1.

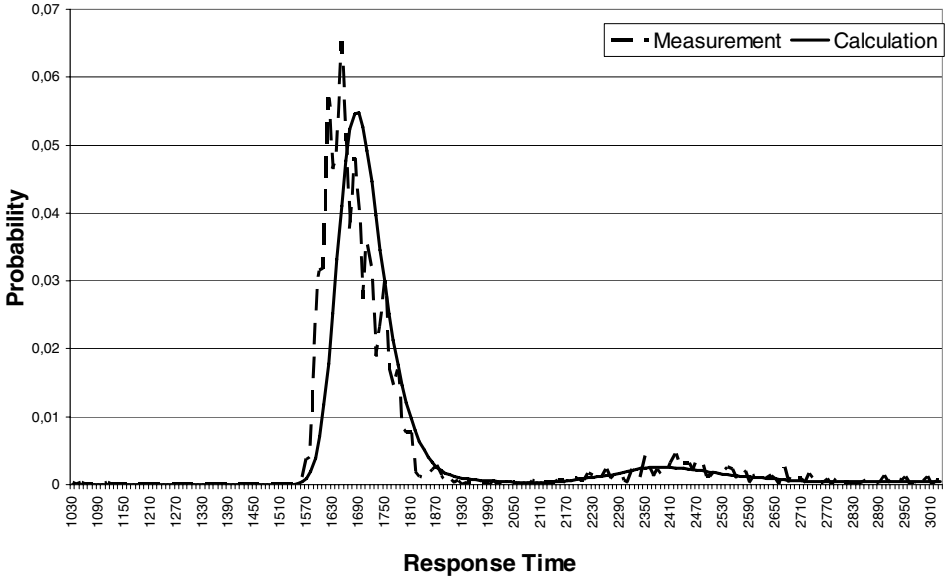


Fig. 10. Scenario 2 (Alternative): Probability Mass Functions

The results for scenario 3 (loop iteration) can be found in Fig. 11. The shape of the lines of measured and calculated probabilities appear similar, although a

| | Mean | Std. Dev. | Max Y |
|---|---|---|---|
| Measurement ($\mu$s) | 2215,38 | 1681,25 | 1650 |
| Calculation ($\mu$s) | 2206,66 | 1508,89 | 1690 |
| Deviation (%) | 0,39 | 10,25 | 2,37 |

Table 2
Scenario 2 (Alternative): Descriptive statistics

deviation can be detected as the measured curve is slightly shifted to the right. As above, the $\chi^2$-test lead to a rejection of the hypothesis of the same underlying probability function.

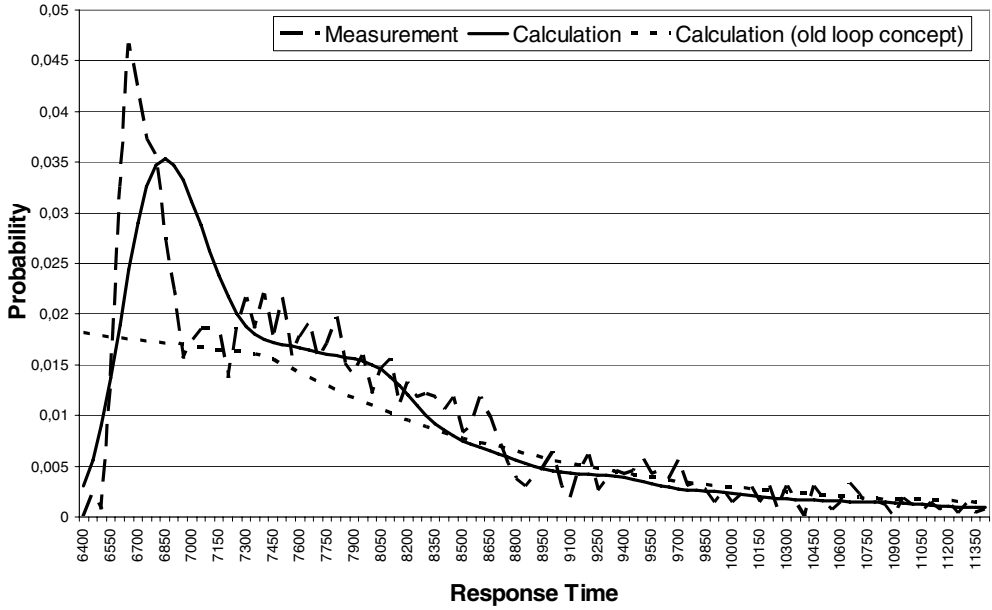


Fig. 11. Scenario 3 (Loop): Probability Mass Functions

The mean values for measurements and calculations only deviate by 1,58 %, the standard deviation and the response time with the highest probability are very similar for both measurements and calculations (Tab. 3). More than 81 percent of the probability values can be found between 6400 $\mu$s and 8800 $\mu$s for the measured values as also for the calculated values. The precision appears useful for performance predictions again.

In Fig. 11, we have also included the calculations for this scenario with a Markovian loop concept (dotted line). With this concept, probabilities are specified for re-entering and for exiting the loop. The situation found in scenario 3 of exactly five loop iterations cannot be expressed. As a result, the calculation does not resemble the measured values as closely as with our calculations with non-Markovian loop iterations. The curve for the Markovian loop model almost looks like an exponential

| | Mean | Std. Dev. | Max Y |
|---|---|---|---|
| Measurement ($\mu$s) | 8160,87 | 2726,85 | 6650 |
| Calculation ($\mu$s) | 8031,65 | 2918,98 | 6850 |
| Deviation (%) | 1,58 | 7,05 | 2,92 |

Table 3
Scenario 3 (Loop): Descriptive Statistics

distribution shifted to the right. The reason for this is, that by specifying loops with transition probabilities, the number of loop iterations is always bound to a geometrical distribution (the discrete case of the exponential distribution). The practical advantage of modelling loops in a non-Markovian way can be clearly observed in the graph, because the predictions do resemble the measurements closer.

### *3.4.2 Sampling Rate*

To analyse the influence of the sampling rate $\alpha$ from the probability mass functions, we obtained measurements and calculations for scenario 1 with different sampling rates (1, 5, 10, 50). The time consumption for the calculations with the different sampling rates was 103 sec, 7 sec, 3 sec, 2 sec respectively, clearly showing the impact of the sampling rate on the timing efficiency of our approach. Looking at the results (Fig. 12, upper-left: $\alpha = 1$, upper-right $\alpha = 5$, lower-left: $\alpha = 10$, lower right $\alpha = 50$, it can be observed that the calculations closely resemble the measurements up to a sampling rate of 10. Only the results with a sampling rate of 50 differed from the measurements to a greater extent. This shows, that we can perform our calculations with a higher sampling rate without loosing much precision.

### *3.4.3 Complexity and Efficiency*

In the following, we discuss the time complexity and time efficiency of our computations. For calculating the timing behaviour of sequences or loops, convolutions of the corresponding probability mass functions have to be performed. We use discrete Fourier transformations, because the convolution becomes a product in the frequency domain.

First we will analyse the *time complexity* of the calculations. Without loss of generality, consider a random variable $X$ describing the timing behaviour of the loop body. Let $w$ be the number of values of $X$ and let $N$ be the maximal possible number of loop iterations determined by the function $l$. Before the Fourier transformation, the value range of $X$ has to be enlarged to $Nw$. The discrete Fourier transformation of a random variable with $Nw$ values has the complexity of $O(N^2w^2 + Nw)$. The N-fold convolution of the discrete random variable corresponds to the N-fold pointwise product of the Fourier transform. As the Fourier transform also has $Nw$ values, the complexity of the N-fold product is $O(N^2w)$. Afterwards, the inverse Fourier transformation has to be performed, having the same complexity as the Fourier transformation. Altogether, the complexity of the computation of the probability

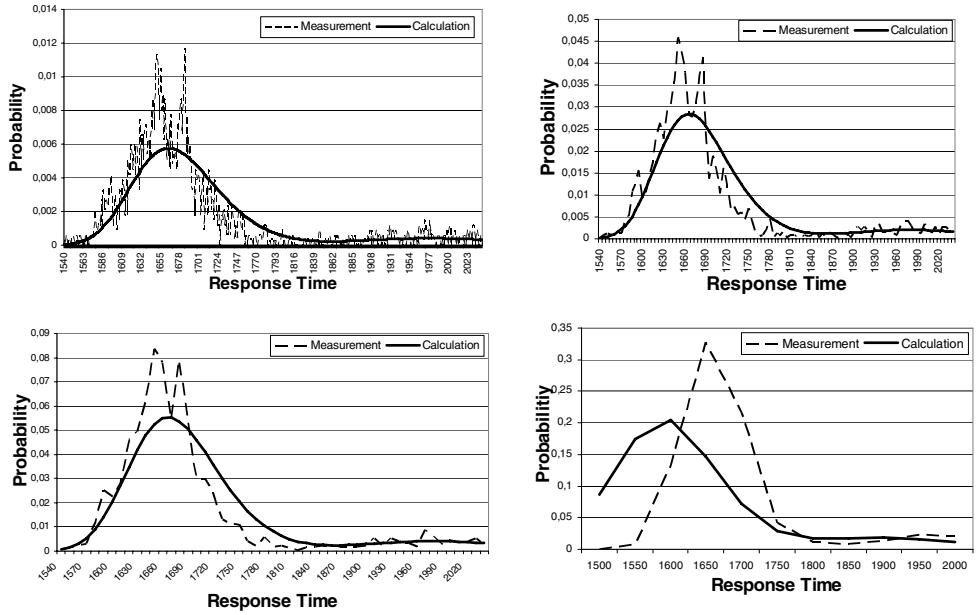


Fig. 12. Influence of the sampling rate

mass function of N-fold loop iterations is $O(N^2w^2 + N^2w + Nw)$.

We also took the time for the calculations of the scenarios described above to analyse the actual *time efficiency* of our approach in our setting. With a sampling rate of 10, the calculations consumed 3 seconds for scenario 1, 7 seconds for scenario 2, and 3 seconds for scenario 3. Additional time was consumed to derive the input data for the calculations from the measurements. For our evaluation, the time efficiency of our calculations was adequate. So far, we have not optimised the code of the calculations, neither have we analysed more complex components.

# 4 Related Work

Our approach aims at supporting design decisions during early life-cycle stages of a software system. The timing and resource efficiency of a component-based software architecture shall be assessed as early as possible during development to avoid the costs of redesign after starting the implementation. The SPE (Software Performance Engineering) methodology by Smith et. al. [17,13] was one of the first approaches into this direction. Balsamo et. al. [18] provide a broad survey on model-based performance prediction methods.

Recently, Becker et. al. [2] specifically classify performance prediction methods for component-based systems. They distinguish between quantitative and qualitative approaches. Quantitative approaches are refined into measurement-based, model-based, and combined approaches. Our performance prediction method is a quantitative approach, because it aims at providing the system designer with performance metrics such as response time or throughput. It is furthermore model-based,

as a special kind of stochastic process is used to carry out the analysis.

A number of other model-based approaches have been proposed. Sitaraman et. al. [19,20] tackle the difficult problem of specifying the performance of a software component. They extend classical O-Notations to specify the time and space requirements of a component and also address the dependency of performance to input data. Hissam et. al. [21] propose a common terminology for Quality of Service prediction of component-based systems. Hamlet et. al. [12] use the information of how subdomains of inputs on provided interfaces are mapped to the interfaces of subsequent components to make performance and reliablity predictions. Bertolino et. al. [11] developed the CB-SPE framework, which is based on the SPE-methodology. In this approach, UML models for component-based architectures are annotated with performance values according to the UML SPT profile [22] and then mapped to queueing networks. Wu et. al. [23] define an XML-based Component-Based Modelling Language to describe the performance of a component-based software system and generate Layered Queueing Networks. Eskenazi et. al. [24] introduce an incremental method to predict the performance of system composed out of components, allowing analytical, statistical, or simulation-based models at each step of the approach. The approach by Chaudron et. al. [25] aims at real-time component-based systems and uses simulations to predict the performance.

Further work is related in terms of the notations used. Parametric contracts for software components have been developed by Reussner et. al. [15]. Stochastically enhanced service effect specifications have been used for reliability predictions for component based architectures [26]. The Quality of Service Modelling Language (QML) [6] can be used to express performance contracts for components. Stocharts [27] are stochastically enhanced statecharts similar to our annotated service effect specifications. However, the number of loop iterations is bound to a geometrical distribution in this approach. The problem of modelling loops with Markov chains is discussed by Doerner et. al. [10]. Stochastical regular expressions have been used for example by Garg et. al. [4]. The SOFA component model uses behavioural protocols similar to our provides and requires protocols to specify allowed sequences of services calls, which are used for interoperability checking [28].

## 5 Conclusions and Future Work

In this paper, we have extended our previous parametric performance contracts [3] by introducing a new loop concept to better model practical situations. We have used stochastic regular expressions instead of Markov models, because it is easier to identify loops and the effort for the calculations is reduced by the ability to traverse the abstract syntax tree of the expressions. Furthermore, we presented an experimental evaluation, finding a close resemblance between measured data and data calculated. We also discussed the efficiency of the approach.

Encouraged by our experimental evaluation, we think that our approach of modelling component performance will be useful for component developers and system assemblers in the future. Architects of component-based systems will be able to

identify bottlenecks in their architectures and will also be supported when choosing among components with equal functionality but different QoS-characteristics.

However, the approach is far from being complete or being applicable in practice right away. Future work aims at including concurrency to be able to model a large class of practical applications. Furthermore, the performance predictions shall be parameterised for the underlying hardware and middleware platform, which is still implicit in our probability function. We also envision a better treatment of the usage profile of components, as so far we assume that the necessary values for usage modeling are already available. The computational complexity as well as the practical feasibility also have to be analysed more in depth. [4]

**Acknowledgements**: We would like to thank Jens Happe and Helge Hartmann for helping us to implement and validate our approach.

## References

[1] Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley (2002)

[2] Becker, S., Grunske, L., Mirandola, R., Overhage, S.: Performance prediction of component-based systems: A survey from an engineering perspective. In Reussner, R., Stafford, J., Szyperski, C., eds.: Architecting Systems with Trustworthy Components. Number To Appear in LNCS. Springer (2005)

[3] Firus, V., Becker, S., Happe, J.: Parametric performance contracts for QML-specified software components. In: FESCA '05: Proceedings of Formal Foundations of Embedded Software and Component-Based Software Architectures Workshop. (2005) 64–79

[4] Garg, V., Kumar, R., Marcus, S.: A probabilistic language formalism for stochastic discrete-event systems. IEEE Transactions on Automatic Control **44** (1999) 280–293

[5] Reussner, R.H., Schmidt, H.W.: Using parameterised contracts to predict properties of component based software architectures. In Crnkovic, I., Larsson, S., Stafford, J., eds.: Workshop On Component-Based Software Engineering (in association with 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems), Lund, Sweden, 2002. (2002)

[6] Frolund, S., Koistinen, J.: QML: A language for Quality of Service Specification. Technical Report HPL-98-1U, HP Software Technology Laboratory (1998)

[7] Sisper, M.: Introduction to the Theory of Computation. PWS Publishing Company (2001)

[8] Reussner, R.H.: Parametrisierte Verträge zur Protokolladaption bei Software-Komponenten. Dissertation, Universität Karlsruhe (2001)

[9] Trivedi, K.S.: Probability and Statistics with Reliability, Queuing and Computer Science Applications. John Wiley & Sons (1982)

[10] Doerner, K., Gutjahr, W.J.: Representation and optimization of software usage models with non-markovian state transitions. Information & Software Technology **42** (2000) 873–887

[11] Bertolino, A., Mirandola, R.: CB-SPE Tool: Putting component-based performance engineering into practice. In Crnkovic, I., Stafford, J.A., Schmidt, H.W., Wallnau, K.C., eds.: Component-Based Software Engineering, 7th International Symposium, CBSE 2004, Edinburgh, UK, May 24-25, 2004, Proceedings. Volume 3054 of Lecture Notes in Computer Science., Springer (2004) 233–248

[12] Hamlet, D., Mason, D., Woit, D.: Component-Based Software Development: Case Studies. In: Properties of Software Systems Synthesized from Components. Volume 1 of Series on Component-Based Software Development. World Scientific Publishing Company (2004) 129–159

[4] Further details on the Palladio project are available at:
http://se.informatik.uni-oldenburg.de/palladio

[13] Smith, C.U.: Performance Solutions: A Practical Guide To Creating Responsive, Scalable Software. Addison-Wesley (2002)

[14] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Element of Reusable Object-Oriented Systems. Addison-Wiley (1995)

[15] Reussner, R.H., Poernomo, I.H., Schmidt, H.W.: Reasoning on software architectures with contractually specified components. In Cechich, A., Piattini, M., Vallecillo, A., eds.: Component-Based Software Quality: Methods and Techniques. Number 2693 in LNCS. Springer (2003) 287–325

[16] Freedman, D., Pisani, R., Purves, R.: Statistics. W. W. Norton & Company (1997)

[17] Smith, C.: Performance Engineering of Software Systems. Addision-Wesley (1990)

[18] Balsamo, S., DiMarco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: A survey. IEEE Transactions on Software Engineering **30** (2004) 295–310

[19] Sitaraman, M., Kuczycki, G., Krone, J., Ogden, W.F., Reddy, A.: Performance specification of software components. In: Proc. of SSR '01. (2001)

[20] Krone, J., Ogden, W., Sitaraman, M.: Modular verification of performance constraints. Technical report, Dept. Computer Science, Clemson University, Clemson (2003)

[21] Hissam, S.A., Moreno, G.A., Stafford, J.A., Wallnau, K.C.: Packaging predictable assembly. In: CD '02: Proceedings of the IFIP/ACM Working Conference on Component Deployment, London, UK, Springer-Verlag (2002) 108–124

[22] OMG, O.M.G.: UML Profile for Schedulability, Performance and Time. http://www.omg.org/cgi-bin/doc?formal/2003-09-01 (2003)

[23] Wu, X., Woodside, M.: Performance modeling from software components. SIGSOFT Softw. Eng. Notes **29** (2004) 290–301

[24] Eskenazi, E., Fioukov, A., Hammer, D.: Performance prediction for component compositions. In: Proceedings of the 7th International Symposium on Component-based Software Engineering (CBSE7). (2004)

[25] Bondarev, E., Muskens, J., de With, P., Chaudron, M., Lukkien, J.: Predicting real-time properties of component assemblies: A scenario-simulation approach. In: EUROMICRO '04: Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04), Washington, DC, USA, IEEE Computer Society (2004) 40–47

[26] Reussner, R.H., Schmidt, H.W., Poernomo, I.H.: Reliability prediction for component-based software architectures. J. Syst. Softw. **66** (2003) 241–252

[27] Jansen, D.N., Hermanns, H., Katoen, J.P.: A qos-oriented extension of uml statecharts. In Stevens, P., Whittle, J., Booch, G., eds.: UML 2003 - The Unified Modeling Language, Modeling Languages and Applications, 6th International Conference, San Francisco, CA, USA, October 20-24, 2003, Proceedings. (2003) 76–91

[28] Plasil, F., Visnovsky, S.: Behavior protocols for software components. IEEE Trans. Softw. Eng. **28** (2002) 1056–1076